**Киевский национальный университет имени Тараса Шевченка**

**Факультет компьютерных наук и кибернетики**

**Алгоритмы и сложность**

**Лабораторный проект № 3  
d-арні піраміди  
Звіт**

**Подготовил:**

студент группы К-28

Шкарупа Микита Олексійович

**Киев-2019**

**Задача алгоритму**

d-арні піраміди схожі на бінарні, тільки їх вузли, відмінні від листя, мають не по 2, а по d дочірніх елементів. Необхідно уявити d-Арну піраміду у вигляді масиву і визначити її висоту для n елементів.

Розробити ефективні реалізації процедур Extract\_Max, Insert і Increase\_Key, призначених для роботи з d-Арно незростаюча пірамідою. Проаналізувати час роботи цих процедур і висловити їх в термінах n і d.

**Вигляд алгоритму**

Піраміда представлена у вигляді масиву mass.  
А вигляд дерева, реалізовано завдяки вірному росположенню елементів у дерева зі своїми індексами, що мають приблизний вигляд b\*i+1,...b\*i+b або ідентично, але зі значенням a.  
Для кожної вершини піраміди ключі всіх її нащадків, менше або дорівнюють власному ключу вершини.

**Функція heapify знаходження головного вузла у купі та її відновлені**

Відновлення починається з отримання індексу верши та аналізом його ключа. Далі ми маємо знайти елемент з найбільшим ключем, тобто вершину. Якщо ключ індексам має максимальне значення, то алгоритм не виконується, бо вершина вже відома. Якщо це не так, то проходимося по іншім індексам, поки не буде знайдено найбільше значення ключа. Поминавши індекс з максимальним ключем та початковий місцями, так що б вершина була першою.  
Складність алгоритму лише полягає З:  
Th (a) = Th (a / b) + b.  
Th (a) = O (b \* log\_ba).

**Функція build що будує купи**

Якщо ми маємо елементу масиву з n-2, то одразу викликаємо для нього heapify, і йдемо далі, поки не пройдемо увесь масив. В результаті ми відновимо властивості для 0-го вузла, таким чином створивши піраміду.  
складність:  
Tb (a) = Σ\_ (i = 1) ^ a 〖Th (i)〗 = O (a).

**Функція Extract\_Max - видалення вузла з максимальним ключем**

З минулого алгоритму, точно відомо, що нульовий індекс дорівнює максимальному ключу. Міняємо нульовий індекс с першим елементи у купі місцями, зменшуємо розмір на один. Викликає heapify для вузла вже нового вузла.  
Складність:Te a = Th (a) = O (d \* log\_ba).

**Функція Increase\_Key - збільшення ключа**

Наступні індекси у купі, викликатися таким чином, через що воно має структуру дерева:  
p = (i - 1) / d.  
Що б замінити ключ, на елемент що має більше значення, алгоритм виканує: прохід по нащадках до самого головного вузла з нульовим індексом, чи не знайдемо індекс вершини в якій виконуються властивості піраміди.   
складність:Tic (n) = Tic (n / d) + 1, Tic (n) = O (log\_d⁡n).

 Вставка  Insert

Якщо треба додати новий елемент, то збыльщуемо розмір купи на один, після чого викликаємо функцію Increase\_Key з новим елментом для останної вершини.  
 складність:  
Tis (n) = Tic (n) = O (log\_d⁡n).